2021

USING TEXT MINING AND MACHINE LEARNING CLASSIFIERS TO ANALYZE STACK OVERFLOW

Taylor Morris
Michigan Technological University, tbmorris@mtu.edu

USING TEXT MINING AND MACHINE LEARNING CLASSIFIERS TO ANALYZE STACK OVERFLOW

By

Taylor Morris

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Computer Science

MICHIGAN TECHNOLOGICAL UNIVERSITY

2021

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Computer Science.

Department of Computer Science

Thesis Advisor:	*Dr. Laura Brown*
Committee Member:	*Dr. Charles Wallace*
Committee Member:	*Dr. Nilufer Onder*
Department Chair:	*Dr. Linda Ott*

Contents

List of Tables

Abstract

StackOverflow is an extensively used platform for programming questions. In this report, text mining and machine learning classifiers such as decision trees and Naive Bayes are used to evaluate whether a given question posted on StackOverflow will be closed or answered. While multiple models were used in the analysis, the performance for the models was no better than the majority classifier. Future work to develop better performing classifiers to understand why a question is closed or answered will require additional natural language processing or methods to address the imbalanced data.

Chapter 1

Introduction

Many programmers turn to the popular forum site, Stack Overflow, when they have a question about programming [1]. Stack Overflow boasts a user base of over 14 million programmers, all working together as a community of experts. The site works by taking in user's questions and allowing other users to answer them. StackOverflow has over 11 million visitors each day. Thanks to the large community of users, a repository of over 21 million user questions and over 31 million answers has built up over time, allowing most to find the answer they are looking for easily [1].

Stack Overflow works by allowing a user to post a question publicly to the site, labeling it with relevant tags, such as the programming language, e.g., "Java" or "Python", or topics / concepts like "sockets". Other users then are able to review

the question and post an answer if they believe they know the solution. The asker can then chose an "accepted answer" when a solution is posted that answers their question best.

In order to prevent poor quality postings, Stack Overflow has a voting system. If a user believes a question or answer is below minimal quality, they can "downvote" the posting. If the user instead finds the posting helpful, or has the same question, they can "upvote" the posting. These "upvotes" and "downvotes" are added together to determine a rating. If a question or answer gets too low a rating it may be closed or even removed.

When a programmer has a question which hasn't been posed before, this system can be daunting. Questions are closed for not being specific enough or for being too similar to another, among other reasons. Additionally, even if a question isn't closed, it could take weeks or even years to get a response.

Users new to Stack Overflow can have questions closed or deleted several times before they are able to make a post that is of high enough quality for the rest of the user base. If they aren't quick to read the comments, they may even miss why their question was removed in the first place.

When time is a valuable resource, as it often is in the programming world, it is desirable to have some ability to predict whether or not your question will be answered

or closed. If the ability were available to developers to check whether or not their question is likely to be answered when submitting a question to StackOverflow, such as in the editor itself, it would allow for users to edit their question or approach to achieve a more desirable result.

Text mining can help with this problem. By taking the vast repository of previous questions and answers available on Stack Overflow and organizing them using text mining approaches, large amounts of data about questions can be analyzed using machine learning methods.

In this report, I will use data from the StackExchange data dump [2] to predict whether a question will be closed or answered. The data is preprocessed into a term-frequency inverse document frequency form, and then run through both CART decision tree and Naive Bayes models. The results showed the insufficiency of the term-frequency inverse document frequency form for this problem.

Chapter 2 discusses the background of text-mining, natural language processing, classifiers, decision trees, and Naive Bayes. Chapter 3 goes over the methods used to produce results. Chapter 4 contains the results and discussion.

Chapter 2

Background

In this report, Stack Overflow questions will be examined using decision trees and Naive Bayes to predict whether a question will be answered or closed.

2.1 Stack Overflow

The average programmer today is familiar with Stack Overflow [1], a forum designed for asking and answering programming related questions. Stack Overflow is open to anyone whether they be professional, a hobbyist, or just starting out. Due to its open nature, posts on Stack Overflow cover a large variety of questions over a number of languages. A quick Internet search for a programming issue is likely to result in a

Stack Overflow question as a top result.

Thanks to the ubiquitous nature of Stack Overflow, analysis on the site offers a way to obtain a large amount of information on programmers' practices and problems. Stack Overflow themselves have analyzed and posted about the patterns of usage. For instance, the analysis looks at posts versus time of day (adjusting for local time) including variables for different languages or topics [3]. Another analysis looks at student usage by programming language, university, time of day, etc. [4].

One prior analysis of Stack Overflow utilized an alternate method of parsing and tagging of data, using an adverb-verb formulation instead of the traditional noun and verb word bagging [5]. This method of analysis was performed on the topics of Stack Overflow posts and allowed the researchers to evaluate the types of questions seen corresponding to different programming languages and different Stack Overflow tags. This analysis shed light on what types of questions are more common between different languages; additionally, this study helped to reveal some of the most common uses of each language [5].

2.2 Text Mining

Text mining is the process of extracting information from text data to solve a specific problem [6]. With the growth of the Internet and the large amounts of text data contain within it, text mining has become incredibly important in recent years.

In order to extract information from text, the text must first be transformed into a more manageable form. This often entails cleaning the text to remove non-relevant terms and combine like terms, in addition to turning the text into a computer understandable structure [7].

One method of structuring the text data uses a structure called a Term-Document Matrix. The Term-Document matrix is constructed by counting how many times a given word occurs in each document. This form of structuring the data is commonly known as the "bag-of-words" form, as it ignores the location of terms within documents [8]. The term-document matrix is created by finding the word occurrence in each document, or $TF(t, d)$. In Term-Document Matrices, the $TF(t, d)$ for each document d and term t decides the value in the t, d entry of the Term-Document Matrix.

Unlike the Term-Document Matrix, Term-Frequency Inverse-Document-Frequency (TFIDF) form accounts not only for the frequency of words within documents but

also for frequency of words across documents. The TFIDF form is based on the idea that words which appear only in a selection of documents may have more importance than words which appear in all documents. This is found by using

$$IDF(t) = \log\left(\frac{|D|}{DF(t)}\right) \tag{2.1}$$

to find the IDF value. D is the number of documents, and $DF(t)$ is the document-frequency of term t. This $IDF(t)$ value is then used to find the finalized weight using

$$W_i = TF(t_i, d) \cdot IDF(t_i) \tag{2.2}$$

where $TF(t_i, d)$ is the frequency of term t_i in document d [9].

Other structures also exist, many along similar lines. One such structure is n-grams, which are terms made of multi-word phrases, used in both TFIDF and Term-Document forms [10].

Many uses for text mining exist, including text classification and knowledge extraction which will be expanded upon in this report. Some other usages which we will not discuss include web mining, web structure mining, content extraction, sentiment analysis, and document summarization [7].

2.2.1 Text Classification

One use of text mining is that of text classification. The text classification problem is based upon the categorization of documents, whether they be emails or news articles. Text classification can help with identifying spam emails or differentiating political news from sports news.

The Spambase dataset from UCI is a popular dataset used in text classification. The goal of the classification is to identify an email as spam (1) or not spam (0) [11]. Another commonly used dataset for text classification is the Reuter's dataset, which aims to classify news documents by category. This dataset is also available on UCI [12].

To classify documents, a database of labeled examples must first be created by experts in the domain being classified. Once labeled, the documents are converted into one of the text mining structured and classified using typical classification techniques such as Naive Bayes or Support Vector Machines [13].

2.3 Classification

Many problems in the natural world are those of classification: what group does item A belong to? These problems are made up of data and a set of classes, and the goal is to decide which class each of the data belongs to. [7].

Our given data is of the form $< \vec{X}, c >$ where $\vec{X}$ is a vector of the form $\{x_1, x_2, \ldots, x_n\}$ and c is a member of $C = \{C_1, \ldots, C_{|dom(C)|}\}$.

2.3.1 Decision Tree

Decision Trees are classifiers which partition the data using a directed tree structure. During the classification process, the classifier begins at the root of the tree and works its way down, looking at the relevant x_i to decide which branch to take at each split [14].

The CART decision tree algorithm used for this project is a binary tree, i.e. each non-leaf node splits into 2 other nodes. CART utilizes a custom metric called the "Twoing Criteria". In cases where there are only two classes to be partitioned into, such as the Answered/Not Answered and Closed/Not Closed partitions presented in this paper, the "Twoing Criteria" is equivalent to the GINI index.

Let D be defined as the set of $< \vec{X}, c >$. Let the domain of a given x_i attribute be defined as $dom(x_i) = \{v_{i,1}, v_{i,2}, \ldots, v_{i,|dom(x_i)|}\}$.

Then the GINI index is defined as

$$GINI(C, D) = 1 - \sum_{C_i \in C} (\frac{|c = C_i \in D|}{|D|})^2. \tag{2.3}$$

where $c = C_i \in D$ is the list of all data points for which the class assignment in $< \vec{X}, c >$ is equivalent to specific class C_i in the overall data set D.

The decision tree is split by maximizing GINI gain, which is calculated from the GINI index using

$$GINIGain(x_i, D) = GINI(C, D) - \sum_{v_{i,j} \in dom(x_i)} \frac{|x_i = v_{i,j} \in D|}{|D|} * GINI(y, x_i = v_{i,j} \in S) \tag{2.4}$$

.

After creating a tree, the decision tree will have some branches which are more informative to the learner than others. In many algorithms, pruning is done to allow the tree to go through as few decisions as possible to reach the results, speeding up the classification time and decreasing the amount of memory space the tree needs. Similarly to learning, pruning utilizes a heuristic to determine which branches to keep and which to remove from the tree. To prune a tree, an algorithm would approach

each node and evaluate whether the better heuristic is gotten with or without that node's child nodes, making the node itself into a leaf of the tree if the former is true [14].

2.3.2 Naive Bayes Classifier

Naive Bayes is a statistical classification method based upon Bayes' theorem, a probabilities theorem which relates conditional probabilities.

Let us have A, B be random variables. Then, Bayes' Theorem is described as

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}. \tag{2.5}$$

The Naive Bayes learner utilizes Bayes' Theorem along with an assumption of conditional independence to calculate the most probable class given the features.

For our prediction, we are given a vector $\vec{X}$ and want to find the correct class assignment for $\vec{X}$, $\hat{C}$. The correct $\hat{C}$ should be the C_i with the highest probability given $\vec{X}$

$$\hat{C} = \arg\max_{C_i \in C} P(C_i|\vec{X}). \tag{2.6}$$

However, we don't know what the value of $P(C_i|\vec{X})$ from just the information given

by our original data. So, we use Equation 2.5 to expand

$$\hat{C} = \underset{C_i \in C}{\arg\max} \ \frac{P(\vec{X}|C_i)P(C_i)}{P(\vec{X})}. \tag{2.7}$$

Here is where the assumption of conditional independence comes in. Since we are assuming each x_l is conditionally independent given C, we can now expand this further

$$\hat{C} = \underset{C_i \in C}{\arg\max} \ \frac{\prod_l^n P(x_l|C_i)P(C_i)}{P(x_1, x_2, \ldots, x_n)}. \tag{2.8}$$

Since the denominator is the same for each class, it is typically ignored

$$\hat{C} = \underset{C_i \in C}{\arg\max} \ P(C_i) \prod_l^n P(x_l|C_i). \tag{2.9}$$

Now we see the most simplified form of Naive Bayes class prediction we get by taking products [15].

In practice, we additionally take the log of the Naive Bayes formula to prevent underflow as the probabilities grow small

$$\hat{C} = \underset{C_i \in C}{\arg\max} \ \log(P(C_i)) + \sum \log P(x_l|C_i). \tag{2.10}$$

2.4 Related Works

Knowledge extraction is the problem of gathering previously unknown information from text. Knowledge extraction differs from general text mining in not trying to solve a particular problem, but instead build knowledge based on text. The knowledge being gathered this way may or may not be known ahead of time. This information can be extracted via rules much like regular expressions created by experts but using pre-made rules is tedious. Instead, researchers in the area have learners deduce new rules from those they are given and the text, allowing entire knowledge bases to be automatically generated [16].

Other researchers have investigated the available StackOverflow data in order to perform text mining to determine quality, trends, and to build an information base.

In one paper, prediction of long-term interest and value along with whether or not a question has been sufficiently answered on Stack Overflow was discussed. The authors of this paper saw to analyze questions on Stack Overflow for the quality of knowledge imparted on it, and the value of the questions and answers to programmers [17].

In another related paper, a group mined Stack Overflow for trends related to web development to discover what types of things web developers asked and answered. This paper made discoveries on how web development is currently being used and

shows how Stack Overflow can be of great use in assessing how programming languages and methodologies are growing and changing [18].

Stack Overflow has also been mined in order to enhance IDE (Integrated Development Environment) abilities. This usage of Stack Overflow suggests that quality questions can, in addition to helping the novice or intermediate programmer answer questions, guide the programmer in their task [19].

Chapter 3

Methods

3.1 Dataset Details and Description

The data to be used is from the Stack Exchange data dump, a regularly made dump of all data from the Stack Exchange family of websites. It is available on archive.org for public use [2]. Specifically, the data found in stackoverflow.com-Posts.7z was used for this analysis.

Each data sample had the attributes found in Table 3.1 and represented a single post. Before the NLP could be done, the data needed to be preprocessed to eliminate unneeded data points and fields. In particular, I was only interested in posts of type 'Question' which were posted in 2020. I limited my analysis to posts from 2020 as

Attribute Name	Type	Description
Id	ID	Unique post identification
PostTypeId	ID	Identifies the type of post via integer 1: Question, 2: Answer
ParentID	ID	Indicates the question being answered (only present if PostTypeId is 2)
AcceptedAnswerId	ID	Identifies the "accepted" or best answer to question (only present if PostTypeId is 1)
CreationDate	Date	Initial date of post
Score	Numeric	Indication of Post 'Quality' as determined by user votes
ViewCount	Numeric	Total number of views
Body	String	Body of question or answer, containing the actual question/answer text
OwnerUserId	ID	UserID of the initial creator of post
LastEditorUserId	ID	ID of last user to edit post
LastEditorDisplayName	String	Display name of last user to edit post
LastEditDate	Date	Date of last edit to post
LastActivityDate	Date	Date of last interaction with post
CommunityOwnedDate	Date	Date post was made into community wiki (only on posts in community wiki)
ClosedDate	Date	Date of post closure (only present if post was closed)
Title	String	Short description of post
Tags	String	List of key topics covered by post
AnswerCount	Numeric	Number of answers to question
CommentCount	Numeric	Number of comments on post
FavoriteCount	Numeric	Number of times post was 'favorite'

Table 3.1
Fields available in Posts.xml

Attribute Name	Purpose
PostTypeId	Used to narrow down to only 'Question' post type
Body	Processed and used via NLP
LastActivityDate	Used to determine recency (i.e. from 2020)
ClosedDate	Used to determine whether or not question was closed
Title	Processed and used via NLP
Tags	Processed and used
AnswerCount	Used to determine whether or not question was answered
CommentCount	Used as data attribute
FavoriteCount	Used as data attribute

Table 3.2
Fields Used

programming languages and habits evolve rather quickly over time, and older data wouldn't be as relevant, while still taking extensive time to process.

The attributes used in this project for processing or analysis are given in Table 3.2. Each Stack Overflow question typically contains a single sentence title. Often, a simplified version of the question appears in the title; however, the primary text and background for the coding question appears in the body. The body can contain Markdown or HTML code, including a special tag to allow for code to be included within the question body: `<code>`.

Beyond this basic preprocessing to eliminate non-useful attributes, I also pared down the data to use only 20% of the posts for training and 20% of the posts for testing.

3.2 Data Preparation and Exploration

While much of the Stack Overflow data was structured, the questions and answers were given as unstructured text. So, before any learning could be performed, it was necessary to clean and structure the text data.

To begin, the text data was tokenized in order to be used in a term-document matrix. Next, punctuation needed to be removed. This ended up being an issue, as many programming terms have punctuation within the term (such as java.utils). After examination, the decision was made to remove punctuation but not separate on it.

Once punctuation was removed, the tokens were scanned for stopwords such as 'the', 'a', 'its' and other common terms. Removing these words serves a few purposes. One purpose is to make the dataset going into the learner smaller. This is helpful due to the large amounts of data taking longer times to process. Additionally, removing the stopwords helps to focus the learner, as the stopwords are largely irrelevant to the categorization of a question.

The final technique performed on the text data was utilizing the Porter Stemmer to stem each of the tokens [20]. By stemming the tokens, words with the same root are combined into one token, in effort to create a more accurate representations of like terms and more accurate results.

To further speed up the learning process, sparse terms were removed from the matrix. Additionally, this allows for the learner to have fewer terms (and less operations) to examine, resulting in a faster learning process. When looking over the sparse terms, I found that some sparse terms needed to be kept, as the terms used in this dataset tend to be fairly unique to topic. Without those sparse terms, there was not enough data to analyze. However, without removing any sparse terms, the dataset to be analyzed did not fit in memory in the necessary format. So, sparse terms were removed so that the sparsest terms had 99% sparsity.

After being completely prepared and cleaned, the data was partitioned for training and testing. To partition, the "createDataPartition" function from the "caret" R library was used. This function allowed to separate the data into even splits based on the classification attribute.

The training/test sets were partitioned differently for the closed and answered classification tasks. For the closed task, both training and test sets had the same proportion of closed questions. Similarly, the training and test sets for the answered task had the same proportion of answered questions.

The total dataset of 40% of the 2020 data has 852800 observations. After removing all sparse terms, 931 attributes remain for each observation, including the variables described in 3.2, the tags, and the terms left after performing the data preparation on the questions' body and titles. Terms from the tags, body, and titles were each

considered separately.

About 4.65% of all 852800 observations were closed, and 71.86% of all observations were answered.

3.3 Classification

The first classification done on our dataset aimed to understand whether or not a given question will be closed. To determine whether or not questions had been closed, the closed date attribute was used. We created a binary variable to track closed versus open questions.

To predict whether a question will be answered, I used the answer count field to check whether or not the answer count was greater than 0. Consistent with how the 'Closed' classification was handled, another binary variable was created to track 'Answered' questions.

Once both tasks were prepared, they were run through each learner in turn.

3.3.1 Decision Trees

The CART decision tree algorithm is implemented as part of the rpart R package. In particular, the GINI index was chosen for the rpart partition variable to replicate CART behavior.

Both the closed and answered learners were pruned to different extents to find the most accurate learner. The pruning levels used the complexity variable to determine the extent of the pruning, and both learners were submitted to the same list of complexity values: $\{1\text{e-}5, 2\text{e-}5, 3\text{e-}5, \ldots, 9\text{e-}5, 1\text{e-}4\}$. These complexity levels were selected by looking at the 'printcp' analysis provided by the rpart package. The 'printcp' function provides an analysis of results on the training data at different complexities which includes the rel. error calculated by $1 - RMSE$ where $RMSE$ is the root mean square error. For the closed learner, this rel. error varied between 0.92051 and 1 for the full set of complexities evaluated and between approximately 0.92 and approximately 0.965 for the selected complexity levels. The answered learner had the rel. error vary between 0.83758 and 1.0 for the full set and between approximately 0.845 and approximately 1.0 for the subset. Higher values of rel. error were avoided being used on their own in order to avoid overfitting the data. By selecting this subset, a good range of complexities could be tested without having to test too many different values for complexity.

3.3.2 Naïve Bayes

The Naïve Bayes learner was provided by the e1071 package in R. The e1071 package implementation of the Naïve Bayes learning algorithm makes the assumptions that all attributes are independent given the target class and are part of a Gaussian distribution.

3.3.3 Correcting Methods

The initial run for the data used terms which were highly correlated to the end result, i.e., ViewCount and Score. While these attributes were disqualified in future runs, using these terms resulted in overly simple models, such as a decision tree with no splits or a single split.

Due to the usage of the platform, high scores also typically mean a high quantity or quality of answers, making the score more or less correspondent to whether or not a question is answered. Similarly, questions are only closed when they receive overly low scores, so the score is also correspondent to the closed result.

Higher view counts are correspondent to both, if only because a higher amount of views means a more likely interaction with these elements.

Due to these factors, both these terms needed to be eliminated.

The results before elimination for the Closed decision tree had an error of only 4%, likely related directly to how many questions were closed in total.

Upon eliminating these terms, it was discovered that the sparsity of 99% was insufficient for any amount of information on the documents, as this could mean terms appearing in over 4 thousand of the documents could be eliminated. Due to the highly specialized nature of programming questions, this means the most significant terms tended to be eliminated.

After taking this sparsity and specialized term issue into consideration, the term-frequency document term matrix was processed as a term-frequency inverse-document-frequency matrix. This makes more rare terms have more weight in the final matrix.

3.3.4 Analysis

The analysis ran predictions over a reserved portion of the data - approximately half of the prepared data from the preparation step.

After running the predictions, the results were analyzed based on their accuracy,

specificity, sensitivity, and balanced accuracy.

The majority variable was chosen as the 'positive' answer to each problem. Thus, the 'positive' answer was 'FALSE' for the closed classification problem and 'TRUE' for the answered classification problem.

Sensitivity is defined as the percentage of 'positive' answers correctly classified, i.e., true positive rate. Similarly, specificity is the percentage of 'negative' answers correctly classified, i.e., the true negative rate. The balanced accuracy is the mean of the specificity and sensitivity, meant to find the accuracy correcting for the data proportionality.

Chapter 4

Results and Discussion

The results of the classification based on the TF-IDF revealed the term matrix form is insufficient for training a model to classify into closed or answered classes. Within the closed question classification problem, none of the trained models managed to be as accurate as simply labelling every question 'Open' would have been.

Similar results were seen within the answered question problem, although CART does manage to match the accuracy of labelling each question as 'answered'.

While the overall accuracy tended to be poor, the balanced accuracy taking into account specificity and sensitivity is slightly better than the majority classifier.

For both learners, the specificity is much better than the sensitivity, suggesting the

Model	Acc. (%)	Spec. (%)	Sens. (%)	Balanced Acc.(%)
Majority Classifier	95.35	100.00	0.00	50.00
Naive Bayes	73.76	74.84	51.47	63.16
CART, CP = 1e-5	94.76	99.12	5.27	52.19
CART, CP = 2e-5	94.81	99.19	5.05	52.12
CART, CP = 3e-5	94.88	99.27	4.76	52.01
CART, CP = 4e-5	94.95	99.37	4.47	51.19
CART, CP = 5e-5	95.04	99.47	4.17	51.81
CART, CP = 6e-5	95.16	99.64	3.36	51.50
CART, CP = 7e-5	95.17	99.66	3.16	51.41
CART, CP = 8e-5	95.20	99.71	2.84	51.27
CART, CP = 9e-5	95.24	99.76	2.64	51.20
CART, CP = 1e-4	95.26	99.78	2.54	51.16

Table 4.1
Closed Classification Results

Model	Acc. (%)	Spec. (%)	Sens. (%)	Balanced Acc.(%)
Majority Classifier	71.86	100.00	0.00	50.00
Naive Bayes	53.74	47.59	69.44	58.52
CART, CP = 1e-5	69.70	91.92	12.97	52.44
CART, CP = 2e-5	70.42	93.42	11.67	52.54
CART, CP = 3e-5	71.37	96.12	8.15	52.13
CART, CP = 4e-5	71.50	96.64	7.30	51.97
CART, CP = 5e-5	71.89	98.60	3.67	51.14
CART, CP = 6e-5	71.90	98.72	3.40	51.06
CART, CP = 7e-5	71.91	98.73	3.41	51.07
CART, CP = 8e-5	71.91	98.80	3.25	51.02
CART, CP = 9e-5	71.91	98.80	3.23	51.01
CART, CP = 1e-4	71.87	99.74	6.63	50.20

Table 4.2
Answered Classification Results

majority class is affecting the classification significantly.

It is possible that, with a more even choice of closed/not closed and answered/not answered data, the results would be better.

These results could be due to a number of reasons, but I suspect it is due to the TF-IDF form used for the text mining in this case. TF-IDF and TF matrices work on an assumption of each word being able to be interpreted independently - order and relation of words aren't considered.

Both being closed and receiving an answer can be considered as measurements of quality on StackOverflow. High quality questions will receive answers and low quality questions will be closed.

High quality vs low quality can't necessarily be determined by a bag of words. While better quality questions are more likely to use things like the `<code>` html tag, the use of the `<code>` tag doesn't mean a question will be high quality.

To assess quality, the relation of words likely needs to be considered in some way. This idea is slightly reflected in results, as Naive Bayes - a model that assumes conditional attribute independence given the class performed significantly more poorly than the decision trees. Unlike the independent assumption of Naive Bayes, decision trees have some form of interaction between the attributes as they branch down: attributes may or may not be tested based on the value of another attribute.

Considering language structure and topic-term relevance is likely necessary to get a more accurate model for answered/closed classification problems. StackOverflow, in particular, looks to users to provide things like minimal working examples and

detailed descriptions of the desired results. s

4.1 Future Directions

The primary aim of future directions would be to correct for the potential causes of inaccuracy in the original predictions. Potential directions include other methods of natural language processing, compound attributes, utilizing historical post data, and more balanced data sampling.

First, I would examine other methods of natural language processing (NLP). Since code in general behaves differently than typical languages, perhaps handling the terms found in `<code>` tags differently from the linguistic descriptions of the questions.

A further analysis would look at combining various common attributes to create compound attributes, to account for the idea that a question about "regex" in "Perl" might be more likely to be answered than a question about "regex" in "Java", due to language associations.

Additionally, further ideas in this topic could see how much past post history for a topic affects the question's likelihood to be closed and/or answered, as being a duplicate question can cause the question to be closed. On this same idea, classifying whether or not a question is a duplicate would be an interesting path.

The balance of the data biased the results and should be accounted for in the future. Potential methods include oversampling, where data points of the closed and not answered classes would be duplicated in training, and undersampling, where data points of the not closed and answered classes would be left out of the training data.

References

[1] Stack overflow. `https://www.stackoverflow.com`.

[2] Stack exchange data dump. Stack Exchange, I. `https://archive.org/details/stackexchange`.

[3] What programming languages are used late at night? Robinson, D. https://stackoverflow.blog/2017/04/19/programming-languages-used-late-night/, **2017**.

[4] How do students use stack overflow? Robinson, D. https://stackoverflow.blog/2017/02/15/how-do-students-use-stack-overflow/, **2017**.

[5] Allamanis, M.; Sutton, C. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 53–56. IEEE Press, 2013.

[6] Feldman, R.; Dagan, I. In *KDD*, Vol. 95, pages 112–117, 1995.

[7] Jiawei Han, Micheline Kamber, J. P. *Data Mining: Concepts and Techniques*, The Morgan Kaufmann Series in Data Management Systems; Morgan Kaufmann Publishers: 225 Wyman Street, Waltham, MA, 02451, USA, 3rd ed., 2012.

[8] Tan, A.-H.; others. In *Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases*, Vol. 8, pages 65–70, 1999.

[9] Jing, L.-P.; Huang, H.-K.; Shi, H.-B. In *Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on*, Vol. 2, pages 944–946. IEEE, 2002.

[10] Wang, X.; McCallum, A.; Wei, X. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 697–702. IEEE, 2007.

[11] Spambase dataset. UCI. `https://archive.ics.uci.edu/ml/datasets/spambase`.

[12] Reuters-21578 text categorization collection dataset. UCI. `http://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection`.

[13] Nigam, K.; McCallum, A. K.; Thrun, S.; Mitchell, T. *Machine learning* **2000**, *39*(2), 103–134.

[14] Rokach, L.; Maimon, O. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **2005**, *35*(4), 476–487.

[15] Rish, I. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, Vol. 3, pages 41–46. IBM New York, 2001.

[16] Rajman, M.; Besançon, R. In *Advances in data science and classification;* Springer, 1998; pages 473–480.

[17] Anderson, A.; Huttenlocher, D.; Kleinberg, J.; Leskovec, J. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 850–858. ACM, 2012.

[18] Bajaj, K.; Pattabiraman, K.; Mesbah, A. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 112–121. ACM, 2014.

[19] Ponzanelli, L.; Bavota, G.; Di Penta, M.; Oliveto, R.; Lanza, M. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 102–111. ACM, 2014.

[20] Porter, M. F.; others. *Program* **1980**, *14*(3), 130–137.